УДК 004.9

ДОСЛІДЖЕННЯ ВПЛИВУ РІЗНИХ ТИПІВ ІНДЕКСІВ НА ШВИДКОДІЮ SQL-ЗАПИТІВ У РЕЛЯЦІЙНІЙ СУБД PostgreSQL

Чао Ван,
Булатецька Л.В. (ORCID: 0000-0002-7202-826X),
Волинський національний університет імені Лесі Українки, Луцьк, Україна

INVESTIGATION OF THE IMPACT OF DIFFERENT INDEX TYPES ON THE PERFORMANCE OF SQL QUERIES IN THE RELATIONAL DBMS PostgreSQL

Chao Wang, Bulatetska L.V.
Lesya Ukrainka Volyn National University, Lutsk, Ukraine

Abstract. The growing volume of data in modern information systems increases the need for efficient SQL query processing in relational databases. One of the key optimization mechanisms is indexing, whose effectiveness depends on the index type, data selectivity, and workload characteristics. This study examines the impact of different PostgreSQL index types—specifically B-tree, composite, and functional indexes—on query performance. The aim of the work is to analyze the efficiency of index structures and identify practical indexing strategies that provide significant reductions in query execution time in applied scenarios. The research shows that the use of indexes makes it possible to replace full table scans with more efficient index-based read operations, which substantially accelerates query execution—from seconds to milliseconds. Range queries benefit greatly from the ordered structure of B-trees, composite indexes demonstrate sensitivity to column order, and functional indexes enable the optimization of expressions and predicates. The study concludes that properly designed indexes are a critical factor for improving the performance and scalability of relational DBMS PostgreSQL without the need for hardware upgrades.

Keywords: relational databases, indexing, B-tree index, composite index, function-based index, PostgreSQL, query optimization.

Вступ. У сучасних програмних системах продуктивність роботи з даними визначає якість та масштабованість сервісів. Особливої уваги набуває оптимізація SQL-запитів, оскільки саме вони формують основне навантаження на реляційні бази даних. Одним із найдієвіших механізмів прискорення обробки даних є індексування, яке дає змогу суттєво зменшити час пошуку, фільтрації та з'єднання таблиць.

Попри ефективність, застосування індексів вимагає обережності: вони збільшують обсяг збережених даних, впливають на швидкість модифікації записів та можуть погіршити продуктивність за неналежного вибору структури чи атрибутів. Це робить дослідження їх реального впливу на систему особливо актуальним в умовах постійного зростання обсягів інформації та вимог до швидкодії.

Метою дослідження є практичне обґрунтування ефективності індексування як ключового методу оптимізації запитів у реляційних базах даних та формування рекомендацій щодо його застосування в сучасних інформаційних системах. Для її досягнення було побудовано тестову модель, виконано серію SQL-запитів різної складності та здійснено порівняння їхнього виконання з індексами й без них. Отримані результати дають змогу оцінити доцільність використання індексів і визначити умови, за яких вони забезпечують найбільший приріст продуктивності.

Методологія дослідження. Для забезпечення коректності та відтворюваності експериментальних вимірювань було сформовано контрольоване середовище, у якому єдиним фактором, що впливав на результати виконання запитів, виступала наявність або відсутність індексів. Підготовчий етап включав налаштування серверного середовища, побудову структури бази даних та генерацію штучних, але статистично наближених до реальних наборів даних.

Експеримент проводився на окремій апаратній платформі з такими технічними характеристиками: процесор Intel Core i7-12700, 32 ГБ оперативної пам'яті, твердотільний накопичувач; операційна система – Ubuntu Server 22.04. У системі керування базами даних PostgreSQL були налаштовані ключові параметри, що впливають на механізми кешування, а також активовано розширену реєстрацію операцій введення-виведення. Сховище даних ініціалізовано зі стандартними параметрами сортування та знеособленим набором локалізацій для гарантування повторюваності експериментів.

Дослідження виконано у двох режимах:

- перед кожним вимірюванням проміжна пам'ять системи очищалася для виключення впливу попередніх операцій на результати виконання запитів;
- система попередньо виконувала запит один раз для заповнення кешу, а потім виконувала його ще три рази для вимірювання часу, при цьому фіксувалися усереднені значення часу виконання.

Структуру бази даних сформовано на основі спрощеної моделі інтернет-магазину (рис. 1.), яка містить чотири таблиці, пов'язані первинними та зовнішніми ключами. Така конфігурація дає змогу комплексно оцінити вплив індексування на ефективність виконання вибірок, операцій об'єднання таблиць та розрахунку агрегованих значень.

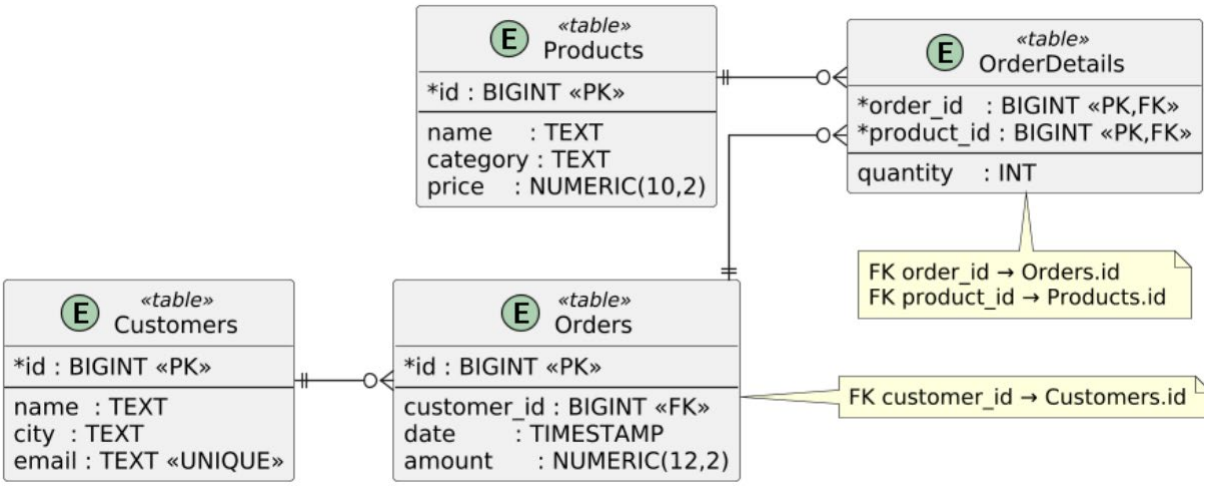

Рис. 1. Логічна схема тестової таблиці (ERD)

Використовувалися такі обсяги даних:

- Customers (Покупці) – 50 000 рядків;
- Products (Товари) – 10 000 рядків у 12 категоріях;
- Orders (Замовлення) – 700 000 записів за період 2022-01-01–2025-10-31 (UTC);
- OrderDetails (ДеталіЗамовлення) – 2 000 000 записів, що пов'язують замовлення з товарами.

Розподіли та значення ідентифікаторів формувалися з урахуванням реалістичності бізнес-даних, що вимагало уникнути рівномірного випадкового вибору. Міста обиралися зі списку з 200 елементів за законом Зіпфа [1], який описує розподіл елементів за частотою появи. Згідно з цим законом, невелика кількість елементів зустрічається дуже часто, тоді як більшість – значно рідше. У контексті нашої бази даних це означало, що невелика кількість міст фігурувала у замовленнях значно частіше, а більшість міст – рідко, що дозволяло моделювати характерні нерівномірності реальних бізнес-даних. Математично закон Зіпфа визначається як обернено пропорційний ранг елемента:

$$f(r) \propto \frac{1}{r^s}$$

де $f(r)$ – частота елемента, r – його ранг, s – параметр розподілу.

Такий підхід забезпечує більш правдоподібний розподіл даних у порівнянні з рівномірним випадковим вибором.

Для вибору категорій товарів використовується фіксований процентний розподіл. Це найпростіший спосіб гарантувати, що «популярні» категорії завжди матимуть більшу частку в базі даних, імітуючи реальність, оскільки кожна категорія товарів отримує заздалегідь визначену ймовірність (відсоток) бути обраною.

Для моделювання цін товарів використовується логнормальний розподіл (μ = 3.6, σ = 0.75) [2, 3]. Це поширений і необхідний вибір для фінансових та цінових даних, оскільки він імітує ситуацію, коли більшість товарів має низьку або середню ціну, але існує невеликий "хвіст" преміальних, дорогих позицій.

Для визначення кількості унікальних товарних позицій в одному замовленні використовується трикутний розподіл [3] (мінімум = 2, максимум = 7, мода = 3). Це простий розподіл, який визначається мінімальним (a), максимальним (b) значенням та модою (c) – найімовірнішим значенням. Це дозволяє точно імітувати поведінку клієнтів. Мінімум (2) забезпечує, що замовлення рідко складається лише з одного товару. Мода (3) встановлює, що три позиції є найпоширенішим розміром замовлення, а обмежений діапазон (2–7) запобігає нереалістично великим покупкам. На відміну від рівномірного розподілу, трикутний розподіл чітко акцентує найімовірніше значення, що значно підвищує реалістичність згенерованих даних.

Перевірки цілісності зовнішніх ключів не виявили порушень, діапазон дат у замовленнях відповідав заданим межам, а середня кількість позицій у замовленні становила 2,86.

Результати та їх обговорення. Для оцінювання продуктивності було застосовано стратегію індексування на основі B- дерева, оскільки вона забезпечує високу ефективність як для запитів за умовою рівності, так і для вибірок за діапазоном [4-6]. Кожний запит тестувався у двох режимах: на неіндексованій схемі та після створення відповідних індексів B- дерев, що дозволило порівняти реальний вплив індексування на час виконання. Експерименти проводилися на сталих параметрах середовища з незмінною конфігурацією PostgreSQL. Для формування достатнього навантаження використовувалися набори даних, що включали 50 000 покупців, 10 000 товарів, 700 000 замовлень і 2 000 000 записів деталей замовлень; такий обсяг гарантував суттєву різницю між повним скануванням таблиць та вибіркою за допомогою індексів. Щоб зменшити вплив випадкових флуктуацій, кожний запит виконувався тричі, після чого розраховувалися усереднені значення часу.

Кожен тестовий запит відображав типові шаблони доступу в корпоративних системах, від простого пошуку до складних аналітичних з'єднань. Час виконання вимірювався командою EXPLAIN (ANALYZE, BUFFERS) та усереднювався за три послідовні запуски. Базовий сценарій (без індексів) порівнювався з оптимізованою схемою, що містить індекси B-дерево на ключових атрибутах (електронна пошта, дата та зовнішні ключі).

Підсумкові кількісні результати продуктивності виконання запитів подано в табл.1. Додавання індексів кардинально змінило стратегію планувальника запитів PostgreSQL [7]. Без індексів кожен запит вимагав повного послідовного сканування таблиці, що спричиняло значні операції вводу/виводу. Пошуки за рівністю отримали найбільшу користь, досягнувши прискорення понад чотириста разів завдяки прямому доступу до індексів. Діапазонні запити також продемонстрували значне покращення, скоротивши час обробки з кількох секунд до мілісекунд. Навіть операції агрегації та

об'єднання показали помітне підвищення продуктивності, оскільки індекси дозволили PostgreSQL мінімізувати кількість зчитуваних сторінок диска.

Таблиця 1

Середня продуктивність запитів до та після застосування індексів індексування на основі B-дерев

Тип запиту	**Умова**	**Час виконання (мс)**		**Приріст (×)**	**План виконання**
		Без індексу	**З індексом**		
Пошук за рівністю	email = 'user5000@example.com'	1425	3.2	445×	Сканування індексу (idx_customer_email)
Запит по діапазону	date BETWEEN '2023-01-01' AND '2023-01-31'	5860	127	46×	Сканування діапазону індексу (idx_order_date)
З'єднання за зовнішнім ключем	orders.customer_id = customers.id	9020	1320	6.8×	Вкладений цикл + Пошук індексу
Групування за категорією	GROUP BY products.category	4870	2280	2.1×	Сканування купи бітмап + Хеш-агрегація
Сортування / розбиття на сторінки	ORDER BY date DESC LIMIT 50	2740	18	152×	Сканування індексу (idx_order_date_desc)

Ефективність індексів пояснюється їхньою ієрархічною структурою (B-деревом). Наприклад, структура індексу на стовпці Orders(date) демонструє, як кореневий вузол зберігає межі ключів, спрямовуючи пошук до проміжних (внутрішніх) вузлів, які, у свою чергу, вказують на листові вузли (leaf nodes), що містять фактичні ключі та вказівники рядків (ROWID). Під час виконання запиту, механізм бази даних проходить це дерево від кореневого вузла до потрібної сторінки, отримуючи лише необхідні рядки за їхніми фізичними адресами, замість сканування всієї таблиці.

Індекс B-дерева суттєво зменшує обсяг дискових операцій, особливо у великих таблицях, де повне сканування потребувало б тисяч сторінок. Навіть за мільйонів записів глибина дерева зазвичай не перевищує трьох–чотирьох рівнів, тому пошук вимагає лише кількох операцій доступу. Це пояснює високу ефективність, підтверджену експериментальними результатами.

Отримані дані показують, що індексування забезпечує практичне прискорення виконання запитів: час обробки скоротився із секунд до мілісекунд. Найбільше покращення спостерігається для вибіркових запитів, фільтрів діапазону та операцій сортування. Хоча прискорення для агрегацій та об'єднань є меншим, воно все одно достатнє, щоб обґрунтувати створення індексів на зовнішніх ключах та полях групування.

Наступний етап дослідження був спрямований на вимірювання практичної ефективності різних типів індексів у PostgreSQL. Було протестовано два сценарії: базовий, що містив лише первинні та зовнішні ключі, та оптимізований – із B-деревами, складеними й функціональними індексами. Запити виконувалися також у умовах низької вибірковості, щоб проаналізувати вплив індексів на плани виконання при різних навантаженнях. У базовій конфігурації PostgreSQL використовував послідовні сканування, що спричиняло значні витрати часу та вводу-виводу, і це стало контрольним орієнтиром для порівняння.

В оптимізованій конфігурації були створені індекси B-дерево, складені та функціональні індекси, після чого продуктивність оцінювалася через EXPLAIN (ANALYZE, BUFFERS) із однаковими умовами запуску. Порівняльні результати показали, що індекси B-дерев найбільш ефективні для вибіркових запитів, складені індекси значно прискорюють багатостовпцеві фільтри, а функціональні дають суттєвий

виграш при використанні виразів у предикатах. Наприклад, індекс на upper(name) скоротив час виконання з 1,4 с до 24 мс, а індекс для date_trunc('month', date) — з 2,1 с до 0,1 с. (табл. 2).

Таблиця 2.

Порівняння між базовою та оптимізованою конфігураціями

Тип запиту	**Тип індексу**	**Час виконання (мс)**		**При-ріст (×)**	**Plan type**
		Без індексу	**З ін-дексом**		
Пошук за рівністю (email)	B-дерево	1425	3.2	445×	Послідовне сканування → індексне сканування
Діапазонний запит (date)	B-дерево	5860	127	46×	Послідовне сканування → індексне діапазонне сканування
З'єднання (customer_id)	Композитний (customer_id, date	9020	1320	6.8×	Хеш-з'єднання → вкладений цикл + пошук за індексом
Агрегація (category)	Композитний (category, price)	4870	2280	2.1×	Послідовне сканування → бітове сканування купи (Bitmap Heap Scan)
Сортування (date DESC)	B- дерево	2740	18	152×	Сортування → індексне сканування (попередньо впорядковане)
Пошук без урахування регістру (upper(name))	Функціональ-ний	1460	24	60×	Послідовне сканування → індексне сканування
Агрегація по місяціях (date_trunc('month', date))	Функціональ-ний	2150	105	20×	Послідовне сканування → індексне сканування

На рис. 2 показано, як змінилося виконання запиту після індексації. На ньому показано перетворення для діапазонного запиту на Orders(date), де PostgreSQL замінив повне сканування таблиці прямим обходом структури B-дерева.

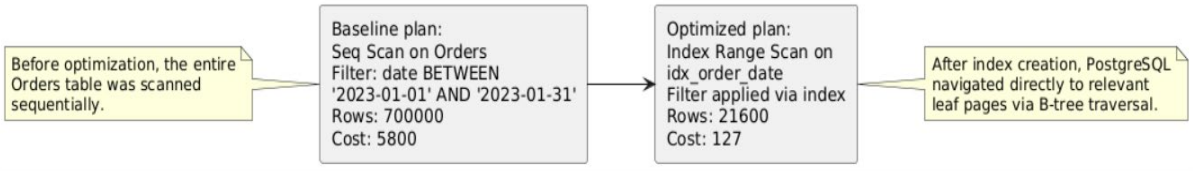

Рис. 2. Трансформація плану виконання після створення індексу

Графічні приклади демонструють перехід від повного сканування таблиці до обходу B-дерева, що суттєво зменшило витрати вводу-виводу та час обробки. Узагальнені результати показали, що проєктування індексів повинно враховувати структуру даних, вибірковість і типи предикатів. Складені індекси виявили чутливість до порядку стовпців, а функціональні усунули неможливість перенесення умов фільтрації на рівень індексу. Загалом експерименти підтвердили, що правильно підібрані індекси радикально покращують продуктивність запитів і оптимізатору значно легше формувати ефективні плани виконання.

Висновки. Результати експериментів показують, що індекси є не допоміжними, а ключовими структурами, які визначають поведінку оптимізатора та межі продуктивності реляційних СУБД. Порівняння базових, B-дерев, композитних і функціональних сценаріїв демонструє, що індекси дозволяють переходити від повного сканування таблиць до вибіркового доступу, суттєво скорочуючи час виконання запитів.

У запитах на рівність та діапазон найбільший виграш забезпечують B-дерева. При високій вибірковості різниця між Seq Scan і Index Scan стає максимальною – від секунд

до мілісекунд. Аналіз складених індексів показав важливість порядку стовпців: коли крайній лівий аргумент відповідає фільтру, індекс використовується повністю; у зворотному випадку ефективність різко падає.

Індекси на основі виразів забезпечили обробку запитів із функціями, які без них примушували до повного сканування. Це підтверджує їхню значимість для реальних систем, де фільтри часто містять обчислення над стовпцями. Поява bitmap scan у сценаріях із низькою вибірковістю демонструє межі індексування: коли дані мають низьку кардинальність, індекс не може повністю усунути витрати вводу-виводу.

У сукупності результати показують, що оптимізація залежить від взаємодії між структурою даних та запитами. Універсального індексу не існує – проєктування має бути ітеративним і ґрунтуватися на аналізі планів виконання. Практичні висновки включають вибір індексованих стовпців за вибірковістю, правильне упорядкування складених індексів та використання індексів на виразах у запитах з обчисленнями.

Отримані дані підтверджують, що продумана побудова індексів може збільшити продуктивність у десятки й сотні разів, підвищити масштабованість та знизити потребу в апаратних ресурсах. Експерименти формують практичну базу для побудови ефективної стратегії індексації та подальшої оптимізації роботи PostgreSQL.

Бібліографія

1. Saichev A., Malevergne Y., Sornette D. Theory of Zipf's Law and Beyond. New York, NY : Springer, 2010. (Lecture Notes in Economics and Mathematical Systems ; vol. 632).
2. Kaldasch J. **Dynamic Model of the Price Dispersion of Homogeneous Goods**. *British Journal of Economics, Management & Trade*. 2015. Vol. 8(2). C. 120–131. DOI: https://doi.org/10.9734/BJEMT/2015/17849
3. Kissell R. L. Probability and Statistics // Algorithmic Trading Methods (Second Edition) : Applications Using Advanced Statistics, Optimization, and Machine Learning Techniques / [Author: Robert L. Kissell]. [Місце видання] : Elsevier, 2021. C. 129–150. DOI: https://doi.org/10.1016/B978-0-12-815630-8.00005-3.
4. Comer D. Ubiquitous B-Tree. *ACM Computing Surveys*. 1979. Vol. 11, no. 2. P. 121–137. URL: https://doi.org/10.1145/356770.356776.
5. 67.4. Implementation. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/16/btree-implementation.html (date of access: 11.12.2025).
6. Postgres Pro Standard : Documentation: 18: 63.1. B-Tree Indexes. *Postgres Professional*. URL: https://postgrespro.com/docs/postgrespro/18/btree (date of access: 11.12.2025).
7. 51.5. Planner/Optimizer. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/current/planner-optimizer.html?utm_source=chatgpt.com (date of access: 11.12.2025).

References

1. Saichev A., Malevergne Y., Sornette D. Theory of Zipf's Law and Beyond. New York, NY : Springer, 2010. (Lecture Notes in Economics and Mathematical Systems ; vol. 632).
2. Kaldasch J. **Dynamic Model of the Price Dispersion of Homogeneous Goods**. *British Journal of Economics, Management & Trade*. 2015. Vol. 8(2). C. 120–131. DOI: https://doi.org/10.9734/BJEMT/2015/17849
3. Kissell R. L. Probability and Statistics // Algorithmic Trading Methods (Second Edition) : Applications Using Advanced Statistics, Optimization, and Machine Learning Techniques / [Author: Robert L. Kissell]. [Місце видання] : Elsevier, 2021. C. 129–150. DOI: https://doi.org/10.1016/B978-0-12-815630-8.00005-3.
4. Comer D. Ubiquitous B-Tree. *ACM Computing Surveys*. 1979. Vol. 11, no. 2. P. 121–137. URL: https://doi.org/10.1145/356770.356776.
5. 67.4. Implementation. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/16/btree-implementation.html (date of access: 11.12.2025).
6. Postgres Pro Standard : Documentation: 18: 63.1. B-Tree Indexes. *Postgres Professional*. URL: https://postgrespro.com/docs/postgrespro/18/btree (date of access: 11.12.2025).
7. 51.5. Planner/Optimizer. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/current/planner-optimizer.html?utm_source=chatgpt.com (date of access: 11.12.2025).